## Section III:

## AMENDMENT UNDER 37 CFR §1.121 to the DRAWINGS

No amendments or changes to the Drawings are proposed.

## Section IV:

## AMENDMENT UNDER 37 CFR §1.121

## REMARKS

**Summary of Telephone Interview**

On April 6, 2005, Examiner Mitchell, Primary Examiner Wei Zhen, and applicant's agent Robert H. Frantz, held a telephone interview at the applicant's agents request in order to consider a proposed amendment and arguments provided by applicant's agent.

Mr. Frantz explained that the proposed amendment incorporated the limitations of Claim 2 into Claim 1 regarding the invention's operation with respect to object-oriented programming executable objects and re-entrant code. As for Claim 1, this should overcome the novelty rejection over Kaneshiro alone, as the Office Action stated that Claim 2 was rejected under 35 U.S.C. 103 over Kaneshiro in view of Orr, wherein Kaneshiro "does not explicitly address object instances" (pg. 4, line 14 - 15) so Orr is cited for that teaching. Mr. Frantz also pointed out that Orr addresses only data objects in the form of engineering change documents (Orr column 1 line 12, col. 2 lines 3 - 21), but is silent as to object-oriented executable program objects, and especially re-entrant code.

Some discussion was had regarding definitions of "object", "re-entrant" and "object oriented programming (OOP)". While the examiners agreed that Kaneshiro was literally silent as to the terms "OOP" or "instantiation", Examiner Mitchell suggested that perhaps Kaneshiro did teach or disclose their invention working with "objects" (col. 5, line 35) when Kaneshiro referred to "constructs" being profiled (col. 2 line 34), albeit there was not any explicit teaching of "object-oriented programming methodology executable objects". Mr. Frantz asked if Examiner Mitchell was officially changing position from what he had stated in the Office Action, and Examiner Mitchell said that it was not an official change in position yet, but he may reconsider. Mr. Frantz suggested that programing "constructs" were typically considered to be code structures such as control structures (e.g. "For-loops", "do-loops", While-endWhile, etc.), and logical structures (e.g. if-then-else, case statements, etc.), but that the concepts and functionality of OOP languages such as Java or C++ go well beyond basic programming constructs.

After further discussion, no consensus was reached regarding whether or not Kaneshiro

inherently taught of OOP executable objects via reference to "constructs", although it was agreed by all that neither Kaneshiro or Orr *explicitly* taught OOP executable objects. Examiner Mitchell pointed out that Kaneshiro discloses "threads" which he interpreted to imply re-entrancy or object instantiation. Mr. Frantz disagreed, and stated that multi-threading was not necessarily inclusive of re-entrancy or object instantation.

Applicant's agent suggested that if the proposed amendment approach was employed in the formal reply, the definitions of these terms would be substantiated in the arguments (e.g. either "construct" includes or does not include OOP executable objects). Examiner's appeared to be agreeable that if the definitions were established in that manner, the amendment may overcome the rejections, depending on the exact wording of the final amendment and the arguments presented.

Mr. Frantz also pointed out that one aspect of the claimed invention that was unique was its ability to dynamically assign unique ID values during run-time of the code as each object is instantiated multiple times. In other words, the inserted stack signing code actually generates the unique ID values and other stack signature values for each object instance during run-time, rather than these values being statically assigned during code development. This aspect allows the behavior of re-entrant code and OOP executable objects to be analyzed based on the stack signatures placed on the stack by our invention because each instance of executable code would have its own unique value. Previous debugging tools, such as that of Kaneshiro, only provide a static module ID, assigned during development or compilation, such that each instance of a particular executable object would still have the same ID values as the other instances of the same executable module. Mr. Frantz suggested that additional claim language specifying the creation of the module ID values *during runtime* might emphasize this inventive aspect of the claimed invention. Examiner Zhen expressed an opinion that this may be anticipated by Kaneshiro's runtime debugging functions, including their "counters and timers" (col. 5, lines 55 - 61). Mr. Frantz pointed out that Kaneshiro's "counters" could not be instance counters of OOP executable objects or re-entrant code modules because Kaneshiro is silent as to OOP executable objects, which returned the discussion to whether or not Kaneshiro's "construct" or "object" disclosure was inherently inclusive of OOP objects or re-entrant code instances.

**Objections to the Claims**

In the Office Action, objections to Claims 14 and 15 were made for their depending on themselves. Examiner correctly assumed that they were meant to be dependent on Claim 13, and proceeded with examination on that basis. Claims 14 and 15 have been corrected by the present amendment. Reconsideration of the objections is requested.

**Rejections of Claims 1 - 6 under 35 U.S.C. §101**

In the Office Action, Claims 1 - 6 were rejected under 35 U.S.C. §101 for being directed to non-statutory matter for failing to include a technological embodiment. Claims 1 - 6 have been amended to clearly specify operations to and on a processing stack located in a computer-readable medium. Reconsideration of the rejections is requested.

**Rejections of Claims 1, 3, 7, and 9 under 35 U.S.C. §102(b) over Kaneshiro**

In the Office Action, claims 1, 3, 7 and 9 were rejected under 35 U.S.C. §102(b) as being unpatentable over US Patent 5,950,003 to Kaneshiro, *et al.* (hereinafter "Kaneshiro"). Claims 1 and 7 are independent claims upon which Claims 3 and 9 depend, respectively.

The present amendment adds steps, elements or limitations not taught by Kaneshiro. In the rationale for rejection of Claims 2 and 8, which are also dependent on claims 1 and 7, respectively, examiner noted that Kaneshiro "does not explicitly address object instances". The present amendment adds steps, elements, or limitations previously found in Claims 2 and 8 to claims 1 and 9, such that anticipation under 35 U.S.C. §102(b) is overcome by reciting re-entrant executable object instances.

Reconsideration of the rejections of Claims 1, 3, 7 and 9 is requested.

**Rejections of Claims 2 and 8 under 35 U.S.C. §103(a) over Kaneshiro in view of Orr**

In the Office Action, claims 2 and 8 were rejected as being obvious over Kaneshiro in view of US Patent 5,191,534 to Orr, *et al.* (hereinafter "Orr"), on the reasoning that Orr taught the step, element or limitation regarding "object instances" that was missing from Kaneshiro.

Orr is taken from non-analogous art as it pertains to management of engineering change documents in a manufacturing environment (col. 1, lines 11 - 27), not to debugging software under development:

> In a typical manufacturing enterprise, a design engineer in a laboratory creates a design change document whenever changes to a product, or any of its components, is necessary. Changes to a design may include item data changes, bill-of-material changes, or reference document changes. When the changes are completed, the design change document along with the actual changes in items and bills-of-material is sent to each manufacturing location that uses the item as either an end product or component of an end product. At a manufacturing location, a manufacturing engineer responsible for the item at the specific plant analyzes the design changes and, if appropriate, makes modifications to the design changes to accommodate the manufacturing process at the plant. These changes are passed on to the shop floor for actual production. (Col. 1, lines 11 - 27, emphasis added)

Orr is silent as to applicability of their system to solving problems or assisting in the development of software, such as debugging re-entrant or OOP executable code. Thus, it would not have been obvious to one skilled in the art of designing software development tools to combine Kaneshiro with a portion of an engineering change document management system.

Additionally, Orr discloses using an "objecting oriented approach" to create a many-to-many relationship between engineering change documents and an item database, and Orr refers to their engineering change documents as "objects":

> These and other objects and advantages are accomplished by the present invention in which an object oriented approach is used to establish many-to-many relationships between engineering changes and items in the master item database. When a design engineering change (EC) needs to be made in an enterprise, the design engineer creates an "EC" object that contains the attributes and functions of a change control mechanism. Every "EC" object is given a unique identifier called an object id which all design changes relating to the engineering change will carry in order to facilitate tracking back to this "EC" object. The "EC" object contains administrative information about an engineering change, such as the designer making the change and the reasons for the change. Similarly, when a manufacturing engineer wants to make changes to the design for his location or wants to create a new

> item to implement some changes at his location, he creates an "EC" object called a "MEC" for manufacturing engineering change. (col. 2, lines 3 - 21, emphasis added)

These engineering change "objects" however are not executable, re-entrant or executable OOP code, but instead are documents. While Orr discloses "instances" of these engineering changes documents (col. 4, line 40), they are not referring to instances of executable code created during run time by an OOP or re-entrant software computing system, but are still merely referring to copies or duplicates of their engineering change documents. Thus, Kineshiro in view of Orr fails to teach all of our claimed limitations.

Additionally, Orr is silent as to manipulation or modification of processor stack contents, likely because it is not directed towards such problems. For this reason, there would have been no motivation to modify Orr to operate on OOP or reentrant executable code objects.

For these reasons, reconsideration and withdrawal of the rejections of Claims 2 and 8 is requested.

**Potential Rejection of Claims 1, 2, 3, 7, 8 and 9 under 35 U.S.C. 103**

With respect to the discussion summarized from the examiner's interview, and with respect to potential rejection of amended claims 1, 2, 3, 7, 8, and 9, the definitions of certain terms should be clarified. The first, most fundamental term employed in the cited art and the applicant's application is the term "object". Our claims specify not just any object, but a *re-entrant executable* object such as an *object-oriented programming* object. Random House Webster's "Computer & Internet Dictionary", Third Edition (1999) by Philip Margolis defines "object" as follows:

> **Object:** Generally, any item that can be individually selected and manipulated. This can include shapes and pictures that appear on a display screen as well as less tangible software entities. In object-oriented programming, for example, an object is a self-contained entity that consists of both data and procedures to manipulate the data.

As such, an "object", absent any explicit re-definition in a

disclosure, can be any individually selectable item. The same reference defines "object oriented" as follows:

> **object oriented:** A popular buzzword that can mean different things depending on how it is used. Object-oriented programming (OOP) refers to a special type of programming that combines data structures with functions to create reusable objects (see under object-oriented programming). Ob- ject-oriented graphics is the same as vector graphics. Otherwise, the term object-oriented is generally used to describe a system that deals primarily with different types of objects, and where the actions you can take depend on what type of object you are manipulating. For example an object-oriented draw program might enable you to draw many types of objects, such as circles, rectangles, triangles, and so on. Applying the same action to each of these objects, however, would produce different results. If the action is Make 30, for instance, the result would be a sphere, box, and pyramid, respectively.

Margolis also defines "object oriented programming" as follows:

> **object-oriented programming:** A type of programming in which programmers define not only the data type of a data structure but also the types of operations (functions) that can be applied to the data structure. In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects. One of the principal advantages of object-oriented programming tech- niques over procedural programming techniques is that they enable pro- grammers to create modules that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes ob- ject-oriented programs easier to modify. To perform object-oriented programming, one needs an object-oriented programming language (OOPL). C + + and Smalltalk are two of the more popular languages, and there are also object-oriented versions of Pascal.

In other words, one can use "object oriented" techniques, such as Orr's many-to-many relationship between electronic engineering change documents with a master item database,

without employing "object oriented programming" (e.g. engineering change documents do not need to include executable software procedures).

Further, one can "individually selectable" program "objects" without those program objects being "object oriented programming instances of executable objects" (e.g. they might be program "objects" from non-OOP programming methodologies or languages), such as Kaneshiro's non-OOP debugging system does.

With respect to Kaneshiro's disclosure of "constructs", this term does not inherently include OOP executable program objects. "Constructs" traditionally refers to program control structures such as While Loops, and logical structures such as If-Then-Else structures (e.g. "conditional statements). OOP and re-entrant executable code objects are not simply alternative "constructs", but reflect a programming paradigm and functionality not provided by such "constructs" alone.

There is a lack of definition of this term in the public sources even though it is widely used, and no other references for this definition have been cited by the examiner, so we must turn to Kaneshiro's disclosure to find its meaning and interpretation, which is consistent with applicant's position:

> Specific **program portions** to be subjected to profiling are **loops, timed regions, and conditionals**. For loop profiling, the execution time and iteration count of a designated loop in a program are measured as profile data. For timed region profiling, the execution time of a designated timed region in the program is measured as profile data. For conditional (conditional branch) profiling, the results of judgment regarding a designated condition in the program are processed to obtain statistical information as profile data. (Col. 1, lines 20 - 29, emphasis added) (a1) Type of information (profile data) collected
>
> ...
>
> (a1) A system of the present embodiment collects profile data at the procedure level and the instruction level **for loops and conditionals**. When a source language (sometimes referred to as an "original source code", "original user code", or "original code") to be profiled is High Performance Fortran (HPF) code, there are included **constructs** specific to the language, i.e., array expressions, WHERE constructs, and FORALL constructs. Inputted HPF codes are interpreted by parser

> processing in step S1 of FIG. 1, so that they are converted into an internal representation (IR: intermediate representation) of the compiler.
>
> Three types of information are collected during profiling - an elapsed time, **invocation and iteration counts**, and a dynamic call tree. For procedures, loops, and timed regions the total elapsed time is measured. Frequency counts are measured for all structures to be profiled to mark **the number of times a statement is executed, the number of times a procedure is executed, the number of iterations in a loop, and the number of times a conditional branch is taken**.
> (Col. 7, lines 54 - 65, emphasis added)
>
> ...
>
> Next, restrictions specific to each of **constructs (structures)** to be profiled, i.e., **region, loop, conditional, and procedure** are listed below.
> (Col. 16, lines 18 - 20, emphasis added)
>
> ...
>
> (b1-4) A data structure is defined for each of **constructs, i.e., loops, arithmetic If, logical if, timed regions, procedures, and caller-callee pairs**, which are to be profiled. Each contains a counter(s) and/or a timer(s), and a unique local ID is assigned thereto.

Clearly from these portions of Kaneshiro's disclosure, their "constructs" are nothing more than the traditional meaning of "constructs", such as conditional logical statements, call-return statements, and control structures, and do not include by re-definition of the term any object oriented programming techniques or functions (e.g. classes, instantiation, etc.).

Kaneshiro's "counts" are clearly counts of operation of their "constructs", which does not include counts of instances of OOP executable objects, including how many times a procedure is "invoked". But, "invocation" is not the same as OOP instantiation, as these terms are well understood in the art to have distinctly different meanings:

> **instantiation:** In programming, instantiation is the creation of a real instance or particular realization of anabstraction or template such as a class of objects or a computer process. To instantiate is to create such an instance by, for example, defining one particular variation of object within a class, giving it a name, and locating it in some physical place.
> (1) In object-oriented programming, some writers say that you instantiate a class to create an object, a concrete instance of the class. The object is an executable file that you can run in a computer.

> (2) In the object-oriented programming language, Java, the object that you instantiate from a class is, confusingly enough, called a class instead of an object. In other words, using Java,you instantiate a class to create a specific class that is also an executable file you can run in a computer.3) In approaches to data modeling and programming prior to object-oriented programming,one usage of instantiate was to make a real (data-filled) object from an abstract object as you would do by creating an entry in a database table (which, when empty, can be thought of as a kind of class template for the objects to be filled in).
> *(Source: www.WhatIs.com information technology glossary)*
>
> **invoke:** To activate. One usually speaks of invoking a function or routine in a program. In this sense, the term invoke is synonymous with call.
> *(Source: www.webopedia.com as redirected from www.WhatIs.com)*

From these definitions, when the verb or action is "instantiation" to created an "instance", as we have employed in our claims, it implies more than just calling a function or routine as in an invocation, but many other functions associated with object oriented programming, as well.

Thus, as Kaneshiro is silent regarding any OOP operations, their disclosure of "invocation counts" does not teach our claimed elements, steps and limitations.

For these reasons, a rejection of claims 1, 2, 3, 7, 8 and 9 over Kaneshiro or Kaneshiro in view of Orr would not be supported by the art.

**Rejections of Claims 4 - 6 and 10 - 15 under 35 U.S.C. §103(a)**

In the Office Action, claims 4 - 6 and 10 - 15 were rejected as being unpatentable over Kaneshiro in view of US Patent 6,161,219 to Ramkumar, *et al.* (hereinafter "Ramkumar"). Claims 4 - 6 depend from Claim 1, and claims 10 - 12 depend from claim 7. Claims 13 - 15 are system claims, in which Claim 13 is the independent claim.

According to the rationale for the rejection, Ramkumar was employed to teach the compilation controls as specified in claims 4 - 6 and 10 - 15. As such, Kaneshiro in view of Ramkumar fails to teach the claim steps, elements, and limitations regarding OOP or re-entrant executable code as previously discussed with respect to the rejections of Claims 1 and 7. For these reasons, the rejections of Claims 4 - 6 and 10 - 15 should be withdrawn.

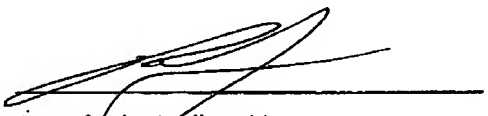
**Presentation of New Claims**

New claims 16, 17, and 18 have been made to steps, elements and limitations regarding encryption of the stack signature to prevent unauthorized access to source code during debug activities. These claims do not add new matter, as their scope is supported by the original disclosure. Consideration and allowance of these claims is requested.

**Conclusion**

The present amendment, and supporting definitions as shown in Appendix A, place the claims in a patentably distinct state over the cited art. The amendment has been revised from the proposed amendment to more clearly present the claimed steps, elements and limitations not taught or suggested by the prior art. Reconsideration of the rejections, and allowance of the claims are requested.

Respectfully submitted,



Agent for the Applicant(s)
Robert H. Frantz, Registration No. 42,553
P.O. Box 23324, Oklahoma City, OK 73123